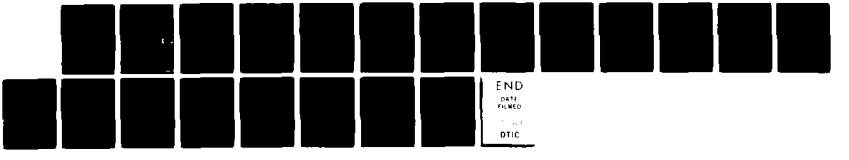
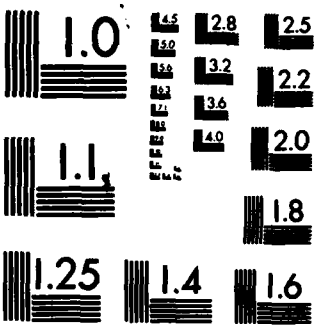
AD-A129 152 PRINGLE: A PARALLEL RROCESSOR TO EMULATE CHIP 1/1
(CONFIGURABLE HIGHLY PARALL..(U) PURDUE UNIV LAFAYETTE
IN DEPT OF COMPUTER SCIENCES J T FIELD ET AL. JAN 83
UNCLASSIFIED CSD-TR-433 N00014-80-K-0816 F/G 9/2 NL



END
DATE FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

BLUE CHiP

Pringle: A Parallel Processor to Emulate CHiP Computers

by

J. Timothy Field
Alejandro A. Kapauan
Lawrence Snyder

The BLUE CHiP Project
Purdue University
Department of Computer Sciences
Math Sciences Building
West Lafayette, Indiana 47907

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CSD-TR-433	2. GOVT ACCESSION NO.: AD-A129152
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): Pringle: A Parallel Processor to Emulate CHiP Computers	5. TYPE OF REPORT & PERIOD COVERED: Technical-Interim
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): J. Timothy Field Alejandro A. Kapauan Lawrence Snyder	8. CONTRACT OR GRANT NUMBER(s): N00014-80-K-0816 N00014-81-K-0360
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Purdue University Department of Computer Sciences West Lafayette, Indiana 47907	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Task SRO-100
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research Information Systems Program Arlington, Virginia 22217	12. REPORT DATE: January 1983
	13. NUMBER OF PAGES: 17
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parallel computer, 64 processors, polled bus, CHiP Computer, MIMD processor, Configurable Highly Parallel Computer, Switch

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Pringle is a 64 processor parallel computer designed to serve as a laboratory instrument for studying Configurable, Highly Parallel (CHiP) Computers. The Pringle's design objectives, architecture and physical characteristics are presented. A key component of the Pringle is a Switch that permits it to emulate CHiP architectures with a variety of corridors widths.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Pringle: A Parallel Processor to Emulate CHiP Computers

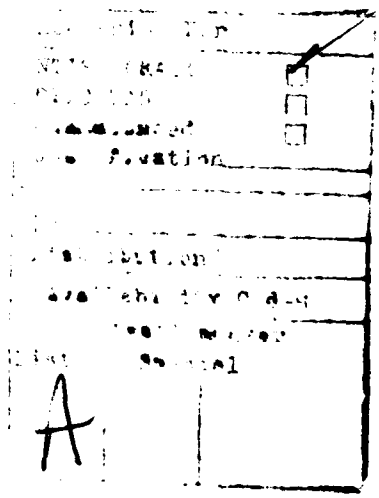
J. Timothy Field
Alejandro A. Kapauan
Lawrence Snyder

Purdue University
Department of Computer Sciences
West Lafayette, IN 47907

ABSTRACT

The Pringle is a 64 processor parallel computer designed to serve as a laboratory instrument for studying Configurable, Highly Parallel (CHiP) Computers. The Pringle's design objectives, architecture and physical characteristics are presented. A key component of the Pringle is a Switch that permits it to emulate CHiP architectures with a variety of corridors widths.

CSD-TR-433



The work described herein is part of the Blue CHiP Project which is funded in part by the Office of Naval Research under Contracts N00014-80-K-0816 and N00014-81-K-0360, the latter is Special Research Opportunities Task SRO-100.

Pringle: A Parallel Processor to Emulate CHiP Computers

J. Timothy Field
Alejandro A. Kapauan
Lawrence Snyder

Purdue University
Department of Computer Sciences
West Lafayette, IN 47907

1. Introduction

The Configurable, Highly Parallel (CHiP) Computers are a family of architectures intended to exploit very large scale integration [1,2]. Because the processors, memory and switching capability compete for the same silicon, there are significant trade-offs possible among the three constituents: large memories imply fewer processors; more switching capability implies smaller processor/memory structures, etc. Determining which family members provide the best mix of these three constituents can only be determined by directly evaluating the needs of the programs written for the CHiP Computer. Software emulation is quickly limited by the low performance that sequential machines exhibit when they emulate multiprocessors. So, a computer to execute CHiP programs is needed in order to design a CHiP Computer. The Pringle serves this purpose.

The Pringle is not a CHiP Computer, but it executes CHiP programs in a way that allows one to infer how a CHiP machine would perform. (This permits software development and testing to proceed in parallel with hardware design.) Moreover, the Pringle is an interesting parallel architecture in its own right.

We begin with a brief review of the CHiP architecture and the design goals for the Pringle (Section 2) and then proceed to describe the Pringle machine in detail (Section 3). We conclude with a comparison of the Pringle and the CHiP Computer.

2. The CHiP Computer and Pringle Design Objectives

Recall from the references [1,2] that a CHiP Computer is composed of a collection of homogeneous processing elements (PEs) placed at regular intervals in a lattice of programmable switches. (See Figure 1.) Each PE is a simple microprocessor with a small amount (e.g., 2K bytes) of local memory for program and data storage; there is *no* global memory. Each switch contains a small amount (e.g., 8-16 words) of memory in which to store switch instructions, called *configuration settings*. Executing a configuration setting causes a switch to connect two or more of its incident data paths; note that this is circuit switching. Separate data paths can cross the switch simultaneously (i.e., there is crossover at a switch). By programming the switches appropriately, the PEs can be connected into topologies of arbitrary form, e.g., mesh, tree, torus. (See Figure 2.)

In addition to the switch lattice, a CHiP architecture has a controlling computer responsible for monitoring the computation. The computation is divided into ***phases***, where each phase corresponds roughly to a single algorithm with a single processor topology. For example, the first phase might be a mesh connected phase, the second a tree connected phase, etc. The controller prepares for a computation by down loading to the PEs the code segments needed for several phases, and down loading to the switches the configuration settings implementing the topologies of those phases. To initiate computation, the controller broadcasts to the

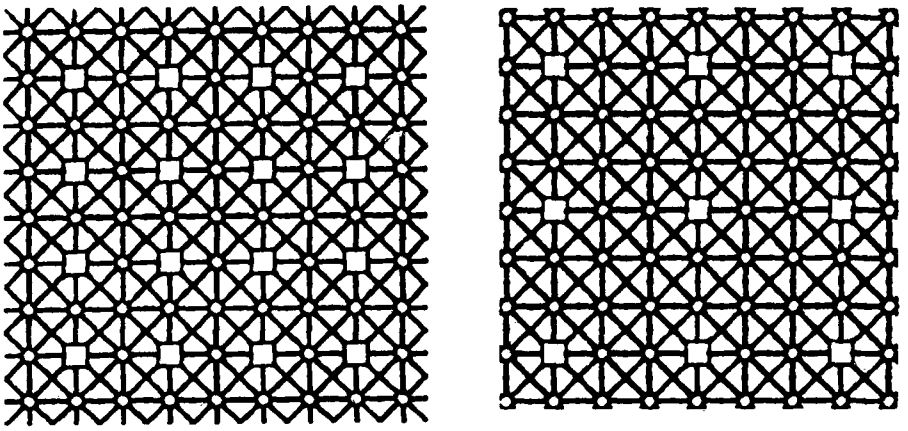


Figure 1. Two switch lattices; squares represent PEs, circles represent switches, lines represent data paths; PEs are actually much larger than switches.

switches which topology is required for the first phase, causing the processors to be connected into that structure. The PEs then begin executing their respective code segments for that phase using a common clock. PEs simply read and write to their I/O parts without "knowing" the source or target PEs of the transfer; the data paths of the configuration form point-to-point connections. When the phase is complete, the controller broadcasts a signal indicating which configuration setting is needed for the next phase, and the PEs then begin executing their corresponding code segments. Execution continues in this manner until the computation is complete or until additional PE and switch codes have to be down loaded.

Although the description of the CHiP machine has been brief, it suffices to permit a discussion of the design goals of the Pringle.

First, we must amplify on a point made in the introduction: The CHiP machine is an integrated architecture intended to exploit VLSI, so the processors, memories, switches and data paths are all competing for the

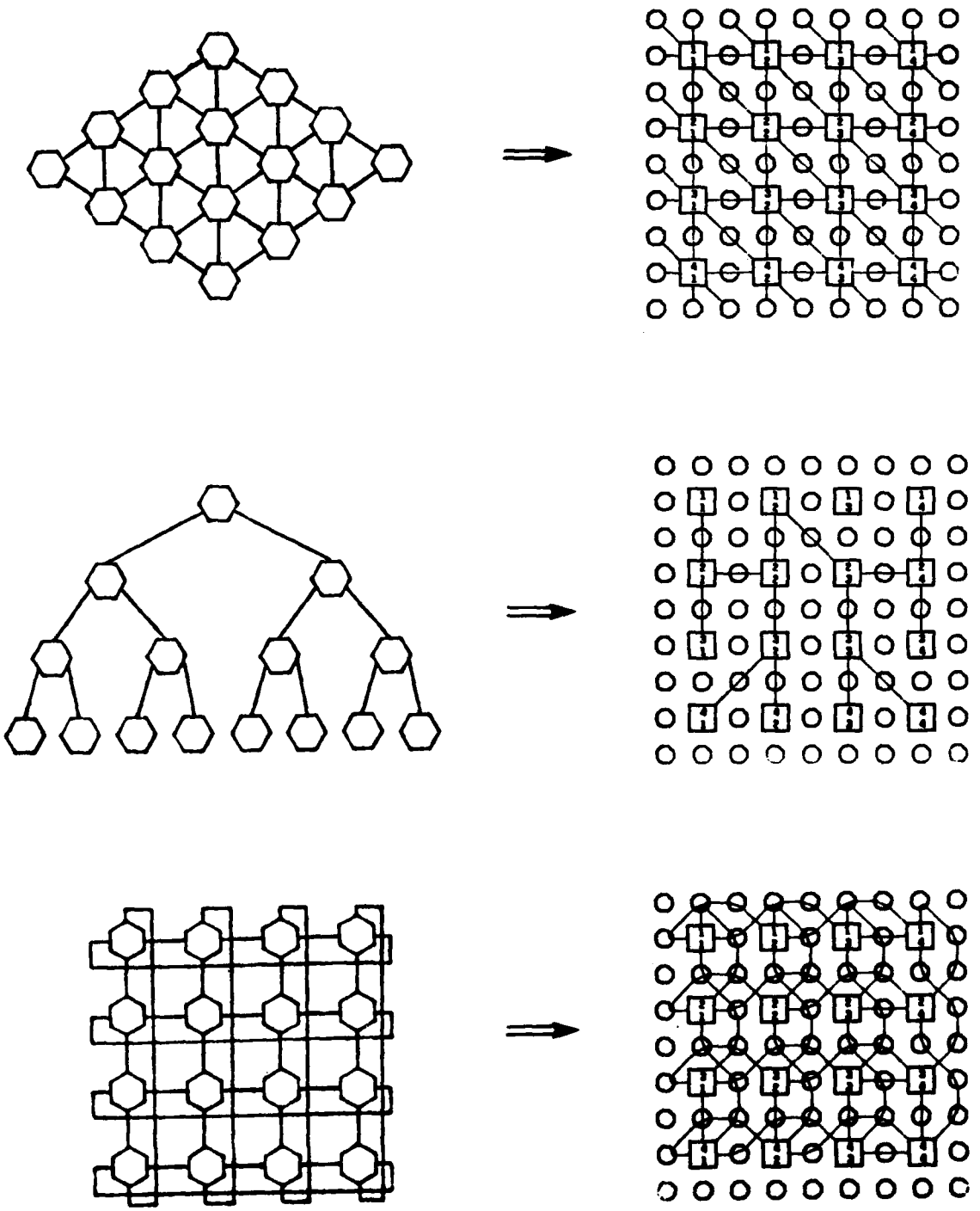


Figure 2. Three configurations of the lattice of Figure 1a.

same resource, silicon. How much area is devoted to each type of component will be determined by how much each contributes to the over all performance of typical algorithms. The answer will be a judgement based on the observed usage. So the Pringle must be a "good, first approximation" with enough flexibility to extend or limit the facilities to some degree.

A second consideration is that the Pringle must have enough processors to test adequately the fine-grain parallelism characteristic of CHiP processing. Of course, no parallel processor has enough processing capacity to handle the largest problems of interest, so we must in any case address the issues of contracting large problems and multiplexing the processors. But there should be enough capacity to observe sustained performance on nontrivial problems.

Another feature of the Pringle design is that it permit a comparison of data driven I/O with synchronous I/O. Data driven communication is expensive to implement because of the need for components like input queues and "overrun" signalling mechanisms. On the other hand, synchronous I/O, which requires PEs to communicate only at agreed upon times, is difficult to program, potentially fragile, but possibly faster. It has been shown [3] that certain data driven programs can be automatically converted into equivalent, synchronously communicating programs. It is crucial to be able to run both to determine the effect on performance.

With these considerations in mind, the Pringle has been designed and built. The structure is described in the next section.

3. Pringle Hardware Description

3.1. Overall System Structure

The Pringle machine was designed with two important requirements in mind: first, the ability to emulate a CHiP computer with reasonable performance, and second, flexibility. Both requirements led to the overall system structure depicted in Figure 3.

The system is divided into two distinct logical parts. The first is a processing element array controlled by a central microprocessor. It contains 64 PEs each of which has its own read-write random access memory (RAM) and read only memory (EPROM). The processors of these PEs are 8-bit single-chip microcomputers coupled with arithmetic processing units (APUs) that perform 32-bit floating point arithmetic.

The PE array is managed by a controller, an Intel 8086, that communicates with the PEs by means of an address-data bus, and a control bus. To facilitate quick down loading of data and programs into PE RAMs from the controller, the RAM of each PE is made to appear as a block of memory in the address space of the controller's microprocessor. This *memory mapping* allows the controller to examine all PE RAM and to take snapshots of memory during the execution of a CHiP program. The control bus includes global reset and interrupt lines that permit the controller to halt or pause the PEs, and status lines which allow the controller to determine the busy or idle status of the PEs.

The second part of the system is the switch lattice emulator, which we shall call the Switch. From a logical point of view, the Switch is a crossbar that allows data transmitted by any PE to be delivered to any other PE, according to routing information stored in a mapping table.

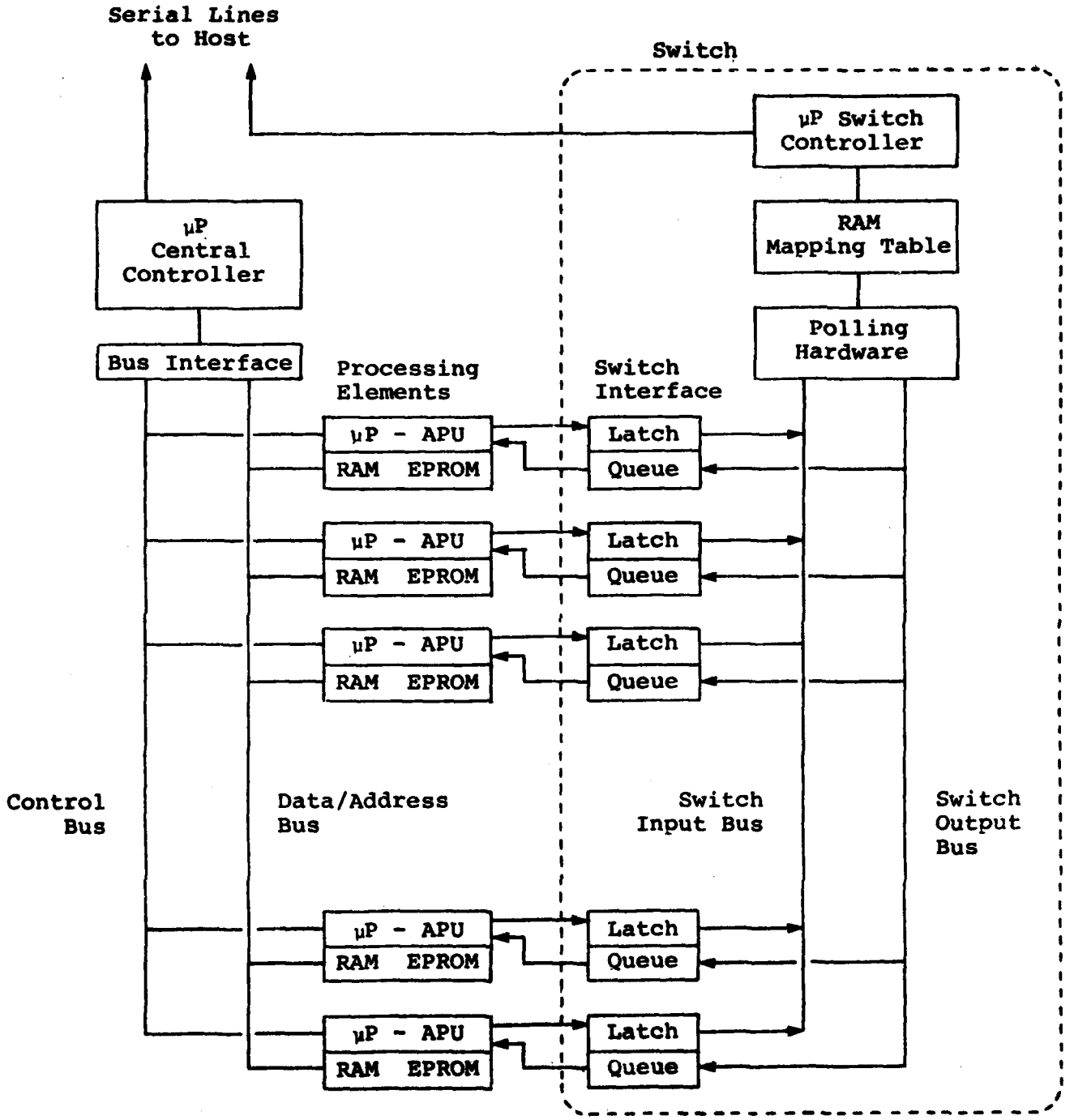


Figure 3. The Pringle Architecture.

The Switch is implemented using high-speed polling hardware.

Every PE is interfaced to the Switch with an output data latch and an input data queue. Assuming processing elements with eight ports as shown in Figure 1, we can assign direction addresses 0 through 7 to the PE I/O ports. When a PE wants to write to an I/O port in Pringle, it latches both the data and the address of the desired direction onto its output data latch, setting a data-present flag on the latch. The Switch polling hardware does a cyclic scan of all the output data latches via the Switch input bus. When it encounters a latch with the data-present flag set, it takes the data and direction number from the latch, clears the data present flag, then looks up in a table in high-speed RAM for the destination PE and port. The polling hardware then routes the data to the input queue of the destination PE with the destination port number appended to the data, via the Switch output bus. The polling hardware will run at a maximum speed of 8 MHz, allowing a complete 64 PE scan to take place in 8 μs. Although not exceptionally fast, this speed is consistent with the computational rate of the PEs.

The RAM which contains the Switch mapping table is accessible to a microprocessor which serves as the Switch controller. It can down load switch settings into the RAM, it can halt and start the polling hardware, and it can detect abnormal conditions in the Switch hardware such as an input queue overflow at one of the PEs.

There is sufficient memory space in the mapping table RAM to hold eight different configuration settings at the same time. This allows up to eight different interconnection structures to be down loaded into the Switch at once. The Switch controller can select any one of the eight configuration settings even as the polling hardware is running.

3.2. PE Structure Detail

Figure 4 presents a block diagram of the PEs implemented in the Pringle machine. The microprocessor used is an Intel 8031, a single-chip 8-bit microcomputer. It contains 128 bytes of internal read-write memory, two parallel I/O ports, two counter-timers, and a serial I/O port. It runs on a 12 MHz clock which gives it a 1 μs execution time for most of its instructions, and a maximum instruction execution time of 4 μs for 8-bit multiply and divide.

External memory is composed of an industry standard 2048 by 8-bit static RAM and a 4096 by 8-bit EPROM. A simple system of tri-state buffers allows the central controller to access the external RAM when it is not being accessed by the 8031.

An Intel 8231 arithmetic processing unit (APU) chip is interfaced to the 8031 by means of a command latch and a data latch. The 8231 contains its own stack to which the 8031 can push data, and from which it can pop data. Commands may be issued by the 8031 to the APU to make it perform floating point arithmetic operations on the stack's contents. As the APU executes commands, the 8031 microprocessor is free to perform other operations.

The 8031 has access to an eleven bit wide output data latch and an eleven bit wide input data queue that, as mentioned earlier, interface it to the switch lattice emulator. Eight of these bits are the data to or received from the Switch, while the other three specify the I/O direction. Since the microprocessor data bus is only eight bits wide, three of the microprocessor parallel I/O port lines serve to extend the data path width to eleven bits. Other I/O port lines serve as control signals to the latch and queue.

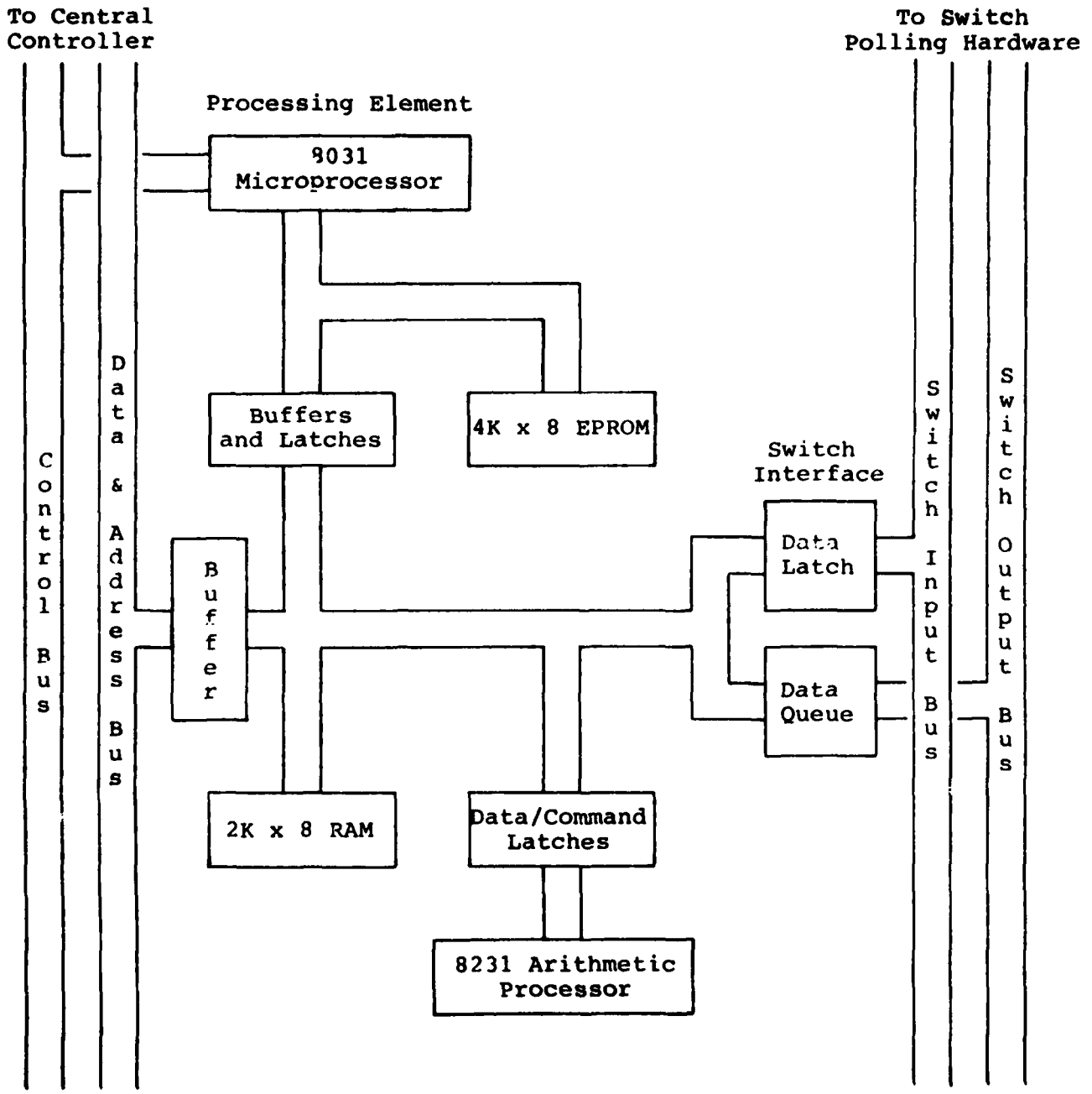


Figure 4. PE Detail.

The input queue acts as a buffer between the Switch hardware, which can operate at a burst data rate of up to 8 MHz, and the relatively slower PE microprocessor. Since all eight logical input ports to the PE are implemented as one physical input port, the use of a buffer queue is essential. The buffer used consists of three bipolar FIFOs, four bits wide by sixteen deep, which produces a single sixteen deep queue. Assuming that the PEs transmit 32 bit words of data, the resulting queue is capable of holding up to four words. This implies that sufficient buffering is present to allow the emulation of CHiP machine programs wherein up to four PEs write to a single PE in a single CHiP machine cycle.

3.3. Switch Emulator Structure Detail

The Switch is implemented in Schottky TTL hardware to permit a very fast clock rate to be used. Figure 5 presents the block diagram of the polling circuitry, mapping table and Switch controller.

A six bit polling counter is used to cycle the input address bus through the addresses of all 64 PEs. When a PE is addressed, it responds by putting the status of its data present flag on the data present bus line, and the contents of its output latch on the data bus. When the hardware detects a latch which contains data, it sends a strobe pulse on a control line on the bus that clears the data present flag of the currently addressed PE, and latches the data on the bus. Using the PE number from the polling counter concatenated with the direction number supplied by the PE, the hardware looks up the mapping table RAM for the destination PE number and direction number.

The destination PE number is latched on the output address bus, and the data from the source PE and the destination port address are latched on the output data bus. Then a strobe pulse is sent on an output control

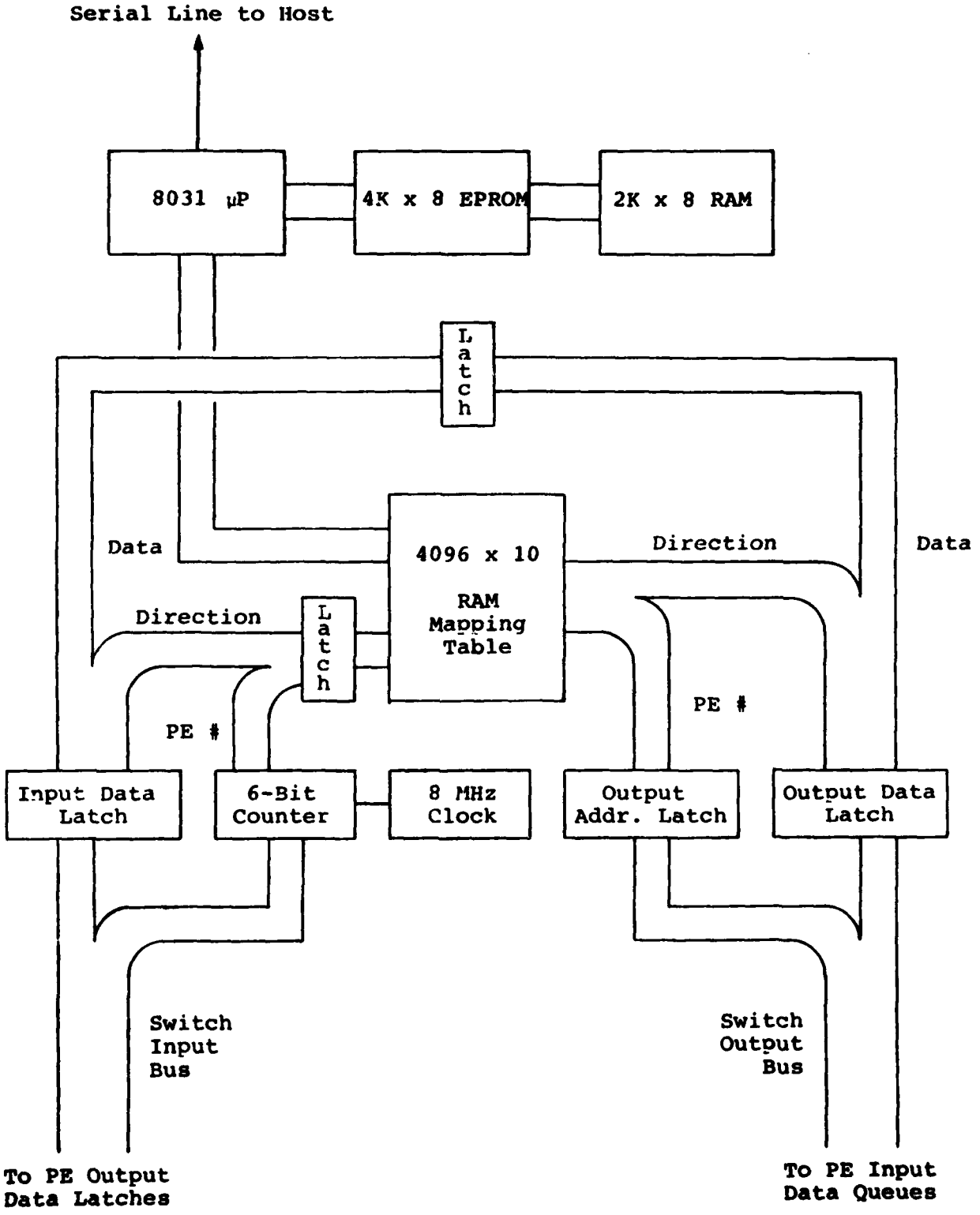


Figure 5. Switch Detail.

line. This causes the contents of the data bus to be entered in the queue of the selected PE.

The entire operation is pipelined to allow the polling of the next input data latch to take place while the data from the current PE is routed to its destination. Notice that this scheme was designed only to emulate a switch lattice for a limited number of PEs. It cannot replace a true switch lattice for an arbitrarily large number of PEs because of the inherent serial bottleneck in sequential polling.

An 8031 microcomputer with 4096 bytes of program EPROM and 2048 bytes of scratch pad RAM serves as the controller of the switch. It can stop the clock on the polling hardware and read or write to the mapping table RAM. By means of three control lines, it can specify which of the eight different switch settings is active at a given instant of time. A serial line allows the 8031 to communicate with the host system.

The mapping table resides in a 4096 by 10-bit word RAM. For each configuration setting, each PE requires eight words, one for the destination PE number and port number of its eight output ports. Thus a total of 512 words are needed per switch setting, giving room for eight different configurations.

3.4. Physical Characteristics

Excluding power supplies, Pringle occupies three 10.5 inch high cages on a standard 19 inch wide rack. Wire-wrap boards are used, 9.6 by 7.8 inches in size, to make hardware modification easy.

There are sixteen PE boards, each containing a cluster of four PEs, and eight Switch boards, each containing the data latches and queues for eight PEs. In addition there is a Switch controller board containing the

polling hardware and control microprocessor for the Switch, and a bus interface board which allows the 8086 central controller to communicate with all the PEs.

Each PE uses 22 ICs; the entire machine, including the Switch, contains 1947 ICs.

4. Comparison of the Pringle and the CHiP Computer

How does the Pringle compare to a CHiP machine? Evidently the 64 Pringle PEs with their 2K RAMs and floating point chips are reasonable, albeit modest, approximations of CHiP PEs. But the Switch bears little relationship to the switch lattice that it is supposed to emulate. This difference warrants further discussion.

Perhaps the most crucial characteristic of a CHiP Computer's switch lattice is the *corridor width*, the number of switches separating two adjacent PEs. (Figure 1 shows two lattices with corridors of width one and two, respectively.) Wide corridors provide greater data routing capability for complex topologies, and although any topology can be embedded into any lattice, those with narrow corridors may underutilize the PEs as a consequence [1,2]. Wide corridors are convenient. On the cost side of the ledger, wide corridors require many switches and data paths, and a reduced proportion of silicon is devoted to processor and memory capacity. Moreover, there is an increased pin requirement per package with wide corridors, and a (minor) increase in transmission delay for neighborhood communication. One central reason for building the Pringle is to determine the best choice for corridor width.

Like all architectural features, the appropriate corridor width is determined by the needs of typical algorithms. Our early algorithmic

experience indicates that a corridor width of perhaps two will suffice for most situations, but much more experience is needed. Any particular choice for the Pringle would have been too inflexible to give this data.

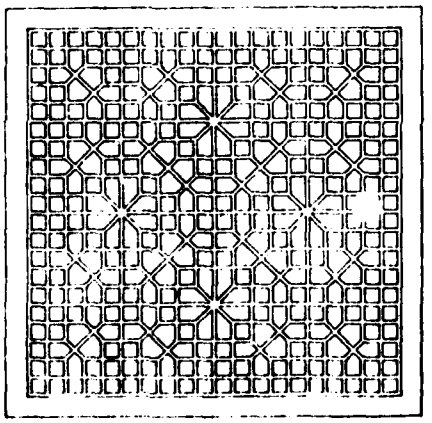
The key to the Pringle Switch's ability to "implement" a variety of lattices rests in the fact that regardless of corridor width, the lattice generally* implements point-to-point communication paths. Thus, a lattice of any corridor width, once reduced to a set of point-to--point communication paths, can be "implemented" by down loading the source-target pairs into the Switch's mapping table.

The routing constraints, imposed on the programmer by a lattice with a particular corridor width, are enforced by the Poker Parallel Programming Environment [4], the Pringle's front end. There the programmer specifies the lattice he wishes to use, programs the interconnection structure graphically, and "compiles" the result into source-target pairs. In Poker it is impossible to violate the limitations of the selected corridor width, so the distilled communication description received by he Pringle is a fair rendering of the routing capability of that lattice. As a result the Pringle looks to the programmer like a CHiP Computer with the lattice of his choice.

*A CHiP switch can fan-out, i.e., broadcast, but this feature can be utilized by other means and has been of only limited utility so far. The Switch cannot broadcast.

Acknowledgements

It is a pleasure to thank J. J. Symanski of the Naval Ocean Systems Center for his guidance and support. D. B. Gannon offered several crucial suggestions that lead to a simpler architecture. Staff of the Purdue Computer Center, especially Lonnie Ahlen, Art De Armond and John Steele have supported us with equipment and advice. John May assisted in a variety of ways with the construction and debugging. Intel Corporation generously donated the processors, memories and floating point chips. We are grateful for all of the assistance.



Buddhist Temple, Mount Omei, Szechwan, 1750 A.D.

References

[1] Lawrence Snyder
Introduction to the Configurable, Highly Parallel Computer
Computer, 15(1): 47-56, January 1982

[2] Lawrence Snyder
Overview of the CHiP Computer
In John P. Gray, editor, *VLSI 81*, pages 237-246, Academic Press, 1981

[3] J. E. Cuny and L. Snyder
Compilation of Data-driven Programs for Synchronous Execution
Proceedings of the 10th *Symposium on the Principles of Programming Languages*, ACM, pages 197-202, 1983

[4] Lawrence Snyder
The Design of the Poker Parallel Programming Environment
Technical Report CSD-TR-409, Purdue University, 1982

END

DATE
FILMED

7-83

DTIC